FigLang

The Natural Programming Language

A language that reads like English
and thinks like a human.

Version 2.1
2025

Table of Contents

1. What is FigLang?

FigLang is a programming language designed from the ground up to be readable, intuitive, and natural. Unlike traditional programming languages that force you to think like a machine, FigLang lets you write code the way you think and speak.

It introduces concepts that no other general-purpose language offers natively: variables with memory, certainty levels, reactive relationships, state machines, contracts, watchers, built-in persistence, a library system, and much more - all expressed in plain English syntax.

FigLang comes with a dedicated VS Code extension providing syntax highlighting, code snippets, hover documentation, and a built-in run button - just like any professional programming language.

```
name is "Marco"
age is 25

if age is at least 18:
    say "Welcome, " and name
otherwise:
    say "Too young"
```

No semicolons. No curly braces. No cryptic symbols. Just English.

2. Why FigLang?

Every programming language today was designed by engineers for engineers. Even Python, often called "readable," still requires learning syntax rules that have nothing to do with how humans naturally express logic.

Consider these everyday tasks:

- Check if a number is between 1 and 100

Most languages: if (x >= 1 && x <= 100)

FigLang: if x is between 1 and 100

- Know the previous value of a variable

Most languages: impossible without manual tracking

FigLang: say previous value of score

- Prevent a variable from going negative

Most languages: write validation everywhere

FigLang: health is never goes below 0

- Validate user input automatically

Most languages: while loop + try/catch + if/else

FigLang: listen for number -> age

- Import reusable code

Most languages: import, require, #include, using...

FigLang: use "library_name"

FigLang was born from a simple question: what if a programming language was designed for humans first, and machines second?

3. How it Works

FigLang is an interpreted language. Your .fig files are processed through three stages:

1. LEXER: Reads your code and breaks it into tokens.

2. PARSER: Takes the tokens and builds an Abstract Syntax Tree (AST).

3. RUNTIME: Walks through the AST and executes each instruction using its own engine with its own memory system, state machine, reactive engine, and contract validator.

Python is only the material used to build the engine, just like C is the material Python itself is built with. FigLang has its own independent systems built from scratch.

4. Getting Started

4.1 Installation

To start using FigLang you need two things:

1. Python 3 installed on your computer.

2. The FigLang files (fig.py, lexer.py, parser.py, runtime.py) in a folder on your computer.

Download FigLang from the official repository and place all files in a single folder. Create a "libs" subfolder for libraries.

4.2 VS Code Extension

FigLang has an official VS Code extension available on the Visual Studio Code Marketplace. It provides:

- Syntax highlighting for .fig files
- Code snippets (type "if" + Tab to auto-complete)
- Hover documentation on keywords
- Run button to execute .fig files directly
- Auto-indentation and bracket closing
- Custom file icon for .fig files

To install:

1. Open VS Code

2. Go to Extensions (Ctrl+Shift+X)

3. Search for "FigLang"

4. Click Install

Once installed, open any .fig file and you will see syntax highlighting, snippets, and the run button automatically.

4.3 Your First Program

Create a file called "hello.fig":

```
name is "World"
say "Hello, " and name
```

```
score is 0
score is 10
score is 25

explain score

if score is above 20:
    say "Great score!"

say current time
say current date
```

Click the play button in VS Code or run from terminal:

```
python fig.py hello.fig
```

That is it. No complex setup, no package manager, no configuration files. Just write and run.

5. Libraries & Imports

FigLang has a built-in library system. Libraries are just .fig files that define zones and aliases. Anyone can create and share libraries.

5.1 Using Libraries

To use a library, simply write:

```
use "game"

player_name is "Marco"
do create_player
do roll_dice
do show_status
```

FigLang looks for the library file in this order:
1. Current folder (game.fig)
2. libs/ folder (libs/game.fig)
The .fig extension is added automatically if missing.

5.2 Creating Libraries

A library is just a .fig file with zones and aliases. Anyone can create one:

```
-- File: libs/greeting.fig

zone called greet_user:
    assume user_name is "Guest" unless defined
    say "Hello, " and user_name and "!"

zone called farewell:
    assume user_name is "Guest" unless defined
    say "Goodbye, " and user_name and "!"
```

Then use it:

```
use "greeting"

user_name is "Marco"
do greet_user
do farewell
```

Output:

```
Hello, Marco!
Goodbye, Marco!
```

5.3 Standard Libraries

FigLang comes with three standard libraries:

math_extra.fig - Mathematical operations: power, abs, max, min, is_even

```
use "math_extra"
base is 2
exp is 8
do power
say result    -- 256
```

strings.fig - String utilities: count_chars, reverse_string

```
use "strings"
text is "hello"
do count_chars    -- 5
```

game.fig - Game development: create_player, damage_player, heal_player, add_score, show_status, roll_dice

```
use "game"
player_name is "Hero"
do create_player
do roll_dice
damage_amount is 25
do damage_player
do show_status
```

Anyone can create and share libraries. Just create a .fig file, place it in libs/, and share it.

6. Core Language

6.1 Variables & Types

```
name is "Marco"
age is 25
price is 9.99
alive is true
friends is ["Alice", "Bob", "Charlie"]
```

No "let", "var", "const", or "=". Just "is". Types are detected automatically.

6.2 Output & Input

```
say "Hello"
say "Hello, " and name
say "You are " and age and " years old"

ask "What is your name?" -> answer
say "Hello, " and answer
```

6.3 Conditions

```
if age is at least 18:
    say "adult"
but if age is between 13 and 17:
    say "teenager"
otherwise:
    say "child"

if name is not empty:
    say "hello, " and name
```

Available: is, is not, is above, is below, is at least, is at most, is between X and Y, is empty, is not empty, contains, starts with.

6.4 Loops

```
repeat 5 times:
    say "hello"
```

```
count from 1 to 10:
    say it

for each friend in friends:
    say "hi, " and friend

until answer is "quit":
    ask "type quit:" -> answer
```

7. Memory System

Every variable automatically remembers its entire history. No setup needed.

```
score is 0
score is 10
score is 25
score is 8

say score                    -- 8
say previous value of score  -- 25
say history of score         -- [0, 10, 25, 8]
say highest of score         -- 25
say lowest of score          -- 0
```

8. Certainty System

Tag values with confidence levels. probably = 80% chance. maybe = 50% chance.

```
speed is definitely 100
temperature is probably 38.5
result is maybe "positive"

if temperature is probably above 38:
    say "might be a fever"
```

9. States

Built-in state machine. No libraries or frameworks needed.

```
door can be open, closed, locked
door starts as closed
door can go from closed to open
door can go from open to closed

door becomes open     -- works
door becomes locked   -- ERROR: cannot go from open to locked
```

10. Reactive Variables

Variables auto-update when dependencies change, like cells in a spreadsheet.

```
price is 100
tax is 20

total reacts to price and tax:
    total is price + tax

say total      -- 120
price is 200
say total      -- 220 (auto-updated)
```

11. Watchers & Events

```
watch score
score is 10    -- [watch] score changed: 0 -> 10
unwatch score

whenever health hits 0:
    say "game over"

every 5 times score changes:
    say "milestone!"
```

12. Contracts, Limits & Assumptions

```
-- Contracts: rules that must be followed
require age to be above 0
require name to not be empty

-- Limits: physical bounds on values
health is never goes below 0 or above 100

-- Defaults: fallback values
assume name is "Guest" unless defined

-- Testing: check conditions
check that age is above 0
check that score is between 0 and 100
```

13. Collections

```
scores is [10, 25, 8, 30, 15]

say average of scores    -- 17.6
say total of scores      -- 88
say highest of scores    -- 30
say lowest of scores     -- 8
say sorted scores        -- [8, 10, 15, 25, 30]
say reversed scores      -- [15, 30, 8, 25, 10]
```

14. String Operations

```
name is "marco rossi"

say name in uppercase         -- MARCO ROSSI
say name in lowercase         -- marco rossi
say name capitalized          -- Marco Rossi
say length of name            -- 11
say name without "rossi"      -- "marco "
say first 5 letters of name   -- "marco"
say last 5 letters of name    -- "rossi"
say name repeated 3 times     -- marco rossi x3
```

15. Natural Math

```
say 10 percent of 200     -- 20
say half of 50            -- 25
say double of 15          -- 30
say square of 4           -- 16
say round 3.7             -- 4
```

16. Unit Conversion

Convert between units naturally. No libraries needed.

```
say 100 celsius in fahrenheit      -- 212
say 10 kilometers in miles          -- 6.2137
say 1024 bytes in kilobytes         -- 1
say 3600 seconds in hours           -- 1
say 90 degrees in radians           -- 1.5708
```

17. Formatting

```
say 1000000 formatted                -- 1,000,000
say 0.5 as percentage                -- 50%
say 3.14159 rounded to 2 decimals    -- 3.14
say 255 in binary                    -- 11111111
say 255 in hexadecimal               -- FF

show friends as list
show scores as bar chart
```

18. Random

```
say random number between 1 and 100
say random item from friends
say random true or false
say shuffled friends
```

19. Files

```
read "data.txt" -> content
say content

write "hello world" to "output.txt"
append "new line" to "log.txt"

lines of "data.txt" -> rows
for each row in rows:
    say row
```

20. Tables

```
table users:
    "Marco"  | 25 | "Italy"
    "Alice"  | 30 | "France"

say row 1 of users         -- [Marco, 25, Italy]
say column 2 of users      -- [25, 30]
```

21. Maps

```
person has:
    name is "Marco"
    age is 25
    city is "Rome"

say name of person     -- Marco
say age of person      -- 25
```

22. Time

```
say current time      -- 14:30:05
say current date      -- 2025-02-26
say current day       -- Thursday

wait 2 seconds

measure time:
    repeat 1000 times:
        score is score + 1
say elapsed time

start timer
stop timer
say timer
```

23. Snapshots

Save and restore the entire program state.

```
score is 100
take snapshot "before battle"
score is 20
restore snapshot "before battle"
say score    -- 100
```

24. Persistence

Save data between sessions. No databases needed.

```
remember score as "my_score"
recall "my_score" -> score
forget "my_score"
```

25. Zones

Reusable blocks of logic.

```
zone called login:
    ask "username:" -> user
    ask "password:" -> pass
    say "Welcome, " and user

do login
do login again
```

26. Pipelines

```
start with numbers, keep above 5, say each

start with numbers
    keep above 5
    double each
    sorted
    say each
```

27. Error Handling

```
try to say unknown_variable
    but if it fails say "variable not found"
```

28. Debug & Explain

```
explain score
-- current value   : 8
-- certainty       : definitely
-- type            : integer
-- changed         : 3 time(s)
-- history         : [0, 10, 25, 8]
-- highest ever    : 25
-- lowest ever     : 0
-- trend           : going down

debug on
score is 50
debug off
```

29. Smart Input (Listen)

Keeps asking until valid input is provided. One line instead of a while loop with try/catch.

```
listen for number -> age
listen for yes or no -> confirmed
listen for one of ["red", "blue", "green"] -> color
```

30. Validation

```
validate email "test@email.com"   -- true
validate email "not_valid"         -- false
validate url "https://google.com"  -- true
validate number "123"              -- true
```

31. Logging

```
log "user logged in"
log "something wrong" with level warning
log "crash!" with level error

save logs to "app.log"
```

32. Compare

```
compare 50 and 80
-- 50 = 50
-- 80 = 80
-- difference: +30
-- 80 is 60% higher

compare [1,2,3] and [2,3,4]
-- in both: [2, 3]
-- only in first: [1]
-- only in second: [4]
```

33. Chain

```
messy is "  hello world  "
clean messy then capitalize then say
-- Output: Hello World

clamp score between 0 and 100 then say
```

34. Aliases

```
alias "greet" means say "Hello!"
do greet

alias "reset" means:
    score is 0
    health is 100
    say "reset complete"
do reset
```

35. Comparison vs Other Languages

FigLang vs Python

Feature	FigLang	Python
Variable	`name is "Marco"`	`name = "Marco"`
Print	`say "hello"`	`print("hello")`
Concat	`say "hi " and name`	`print("hi " + name)`
If	`if x is above 5:`	`if x > 5:`
Between	`if x is between 1 and 10:`	`if 1 <= x <= 10:`
For each	`for each x in list:`	`for x in list:`
Input	`ask "name?" -> n`	`n = input("name?")`
Import	`use "utils"`	`import utils`
History	`say history of x`	`(not possible)`
States	`door can be open, closed`	`(not native)`
Certainty	`x is probably 5`	`(not possible)`
Limits	`x never goes below 0`	`(manual code)`
Watch	`watch score`	`(not possible)`
Explain	`explain score`	`(not possible)`
Validate	`validate email "x@y.z"`	`(regex + code)`
Random	`random number between 1/10`	`random.randint(1,10)`
Convert	`100 celsius in fahrenheit`	`(manual formula)`
Try	`try X but if fails Y`	`try: X except: Y`

FigLang vs JavaScript

Feature	FigLang	JavaScript
Variable	`name is "Marco"`	`let name = "Marco";`
Print	`say "hello"`	`console.log("hello");`
If	`if x is above 5:`	`if (x > 5) {`
Between	`if x is between 1 and 10:`	`if (x>=1 && x<=10) {`
Import	`use "utils"`	`require("./utils")`
For each	`for each x in list:`	`list.forEach(x => {`

History	`say history of x`	`(not possible)`
States	`door can be open, closed`	`(not native)`
Contracts	`require x to be above 0`	`(not native)`
Tables	`table X: row | row`	`(not native)`

FigLang vs Java

Feature	FigLang	Java
Hello World	`say "Hello"`	`public static void...`
Variable	`name is "Marco"`	`String name = "Marco";`
Print	`say "hello"`	`System.out.println();`
Import	`use "utils"`	`import utils.*;`
Loop	`repeat 5 times:`	`for(int i=0;i<5;i++){`
Input	`ask "name?" -> n`	`Scanner sc = new...`
Try/catch	`try X but if fails Y`	`try{X}catch(E e){Y}`
States	`door can be open, closed`	`(enum + switch)`
History	`say history of x`	`(ArrayList + code)`
Logging	`log "msg" with level error`	`Logger.error("msg")`

Features Unique to FigLang

- Built-in variable memory and history tracking
- Certainty system (definitely / probably / maybe)
- Native state machines with transition rules
- Reactive variables (spreadsheet-like auto-update)
- Persistent watchers and event triggers
- Smart input validation (listen for)
- Variable contracts, limits, and assumptions
- Explain command for self-documentation
- Snapshots for saving and restoring state
- Cross-session persistence (remember/recall)
- Native unit conversion
- Built-in data validation (email, url, number)
- Native logging with levels

- Automatic comparison with analysis
- Operation chaining (clean then capitalize then say)
- Custom aliases and macros
- Native tabular data
- Library system (use "library")
- Official VS Code extension
- Natural English syntax throughout

FigLang - Code the way you think.